

NASA/WVU Software IV & V Facility
Software Research Laboratory
Technical Report Series

NASA-IVV-95-004
WVU-SRL-95-004
WVU-SCS-TR-95-24
CERC-TR-RN-95-010

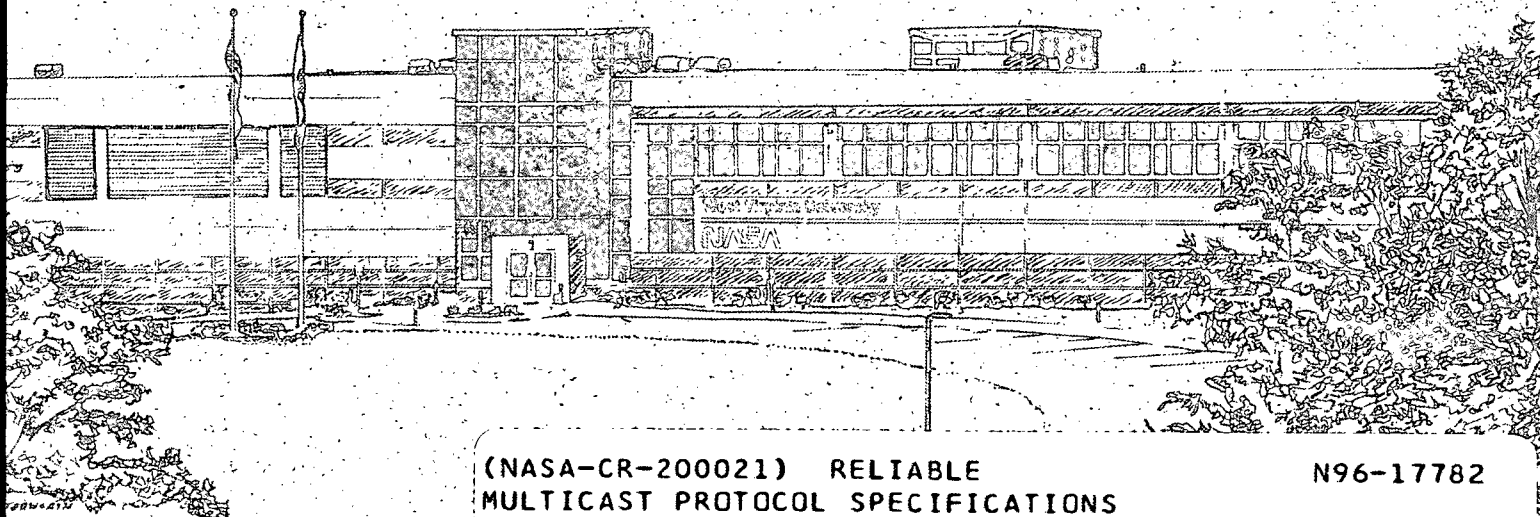
NCCW-0040

*IN-61-CR
7265*

Reliable Multicast Protocol Specifications Protocol Operation

by John R. Callahan, Todd Montgomery, and Brian Whetten

p-22



(NASA-CR-200021) RELIABLE
MULTICAST PROTOCOL SPECIFICATIONS
PROTOCOL OPERATIONS (West Virginia
Univ.) 22 p

N96-17782

Unclas

G3/61 0098253



National Aeronautics and Space Administration



West Virginia University

The Reliable Multicast Protocol Specification
Protocol Operation

RMP Library Version: 1.3b (1.3 Beta)

This appendix contains the complete state tables for RMP Normal Operation, Multi-RPC Extensions, Membership Change Extensions, and Reformation Extensions. First the event types are presented. Afterwards, each RMP operation state, normal and extended is presented individually and its events shown.

Events in the RMP specification are one of several things. (1) Arriving packets, (2) Expired alarms, (3) User events, (4) Exceptional conditions. The specification event types are:

Event Type	Description
Data	Data Packet
ACK	ACK Packet
NACK	NACK Packet
Conf	Confirm Packet
NMD	Non-Member Data Packet
NMA	Non-Member ACK Packet
NL	New List Packet
LCR	List Change Request Packet
RecStart	Recovery Start Packet
RecVote	Recovery Vote Packet
RecACKNL	Recovery ACK New List Packet
RecAbort	Recovery Abort Packet
Failure	Retransmission timeout on packet
TPA	Token Pass Alarm
CTPA	Confirm Token Pass Alarm
RTA	Random Timeout Alarm
MandLv	Mandatory Leave Alarm
CommitNL	Commit New List Notification
JoinReq	Application request to join group

Packet Positive Acknowledgements and Fault Detection

Some packets are retransmitted on a scheduled basis until a given set of condition occur that cease that retransmit schedule. This set of conditions can be considered as a positive acknowledgement for the

packet. The different RMP packets and the positive acknowledgement conditions for each are shown in the table below. The packets with (none) as their positive acknowledgement conditions are only sent once and are not scheduled for any kind of retransmission.

Packet Type	Positive Acknowledgment Conditions
Data	Reception of ACK Packet that contains packet
ACK	Reception of ACK, New List, or Confirm packet with Timestamp $\geq$ Timestamp of packet
NACK	Reception of requested Packet(s)
Confirm	(none)
Non-Member Data (NMD)	Reception of Non-Member ACK Packet that contains packet
Non-Member ACK (NMA)	(none)
New List (NL)	Reception of ACK, New List, or Confirm packet with Timestamp $\geq$ Timestamp of packet
List Change Request (LCR)	Reception of a New List Packet containing response to request
Recovery Start	Creation of a valid New List Packet or X retransmits
Recovery Vote	Reception of a New List Packet from Reform Site
Recovery ACK New List	Reception of Null ACK from Reform Site
Recovery Abort	(none)

If a set number of retransmissions, call this value X, for a packet occur without the positive acknowledgement conditions being met, then a Failure event is generated for that packet. This event is indicative of a failure (or blockage) being detected within the protocol operation (the cause of which being an application failure, a site failure, or a communication failure). Failures in RMP are assumed to be failures that are non-corruptive. All members are assumed to be well behaved in the presence of failures and not miscreant.

Duplicate Detection and Filtering of Packets

Packets must pass through two devices before they are allowed to be processed. The first level is filtering. This level examines the packet type and TRID. If the TRID is not a valid TRID, then the packet is dropped. The state of the site is also taken into account. When the site is in the Not In Ring state or the Joining Ring state, it does not have information about the current TRID of the group. Thus filtering must be done based on packet type and packet contents. For example, as specified below, a site in the Joining Ring state transitions into the Token Site state upon a New List packet. To

filter out possible errant transitions, the filter must drop all other New List packets not meeting the transition criteria.

The second device is duplicate detection. Duplicates are assumed to be dropped and not processed. Duplicates can be detected based on the packet type and state of the site. For packets with explicit timestamps, ACKs and NLs, the timestamp can be used to determine if the packet is a duplicate. For implicit timestamped packets, Data, NMD, and LCRs, the job is slightly more complicated. Careful track of sequence numbers from clients as well as group members must be kept for duplicates to be detected. NMAs, NACKs and Confirm packets are never checked for being duplicates. During reformation, duplicates are allowed to be processed (as long as they are part of reformation. Data, NMD, ACK, NL, and LCR packets are still put through duplicate detection). Another exception to the duplicate detection rule is when the site is in the Token Site state. Duplicate detection is not done on any ACK or NL packets that arrive when the site is in the Token Site state.

Algorithms and Data Structures

Two data structures are used by RMP. The first is the DataQ. This is a FIFO structure that holds Data, NMD, and LCR packets as they arrive. The second structure is an Ordered FIFO called an OrderingQ. This FIFO is ordered based on timestamps. Each member of the FIFO, called a slot, is assigned a unique timestamp. In this way, the FIFO has elements which are monotonically increasing. Each slot of the OrderingQ is one of the following types of packets: Data, NMD, ACK, or NL. ACK and NL packets contain explicit timestamps to order them in the OrderingQ. Data, NMD, and LCR packets are given implicit timestamps by the corresponding ACK or NL that addresses them. Therefore, an ACK with timestamp of 5 which acknowledges 3 packets will implicitly give 3 Data or NMD packets implicit timestamps of 6, 7, and 8. LCR packets are dropped once a corresponding NL is received. LCRs are never put into the OrderingQ.

In the state tables given below, it is assumed that the addition of an ACK into the OrderingQ also enqueues empty slots for each packet acknowledged by the ACK. Each slot in the OrderingQ is assigned a state. There are four possible states for each slot. They are: (1) Packet Missing, (2) Packet Requested, (3) Packet Received, and (4) Packet Delivered. A slot usually starts out as being in the Packet Missing state. Once a NACK is sent for that packet, the state changes to Packet Requested. When the slot is filled by a packet, the state changes to Packet Received, and, finally, when the packet is delivered to the application the slot state is changed to Packet Delivered.

A few algorithms are provided to specify the operation of the OrderingQ and DataQ during the protocol operation. The first algorithm is the Update-OrderingQ algorithm presented below.

```
Update-OrderingQ():
for each (slot in the OrderingQ (starting with lowest timestamp))
  If (slot timestamp not equal to last slot timestamp + 1) then
    EnQueue as many empty slots to cover missing timestamps
    for each (new slot to be Enqueued) Loop
      Send NACK for missing timestamp
      Mark slot state as Packet Requested
    End for.
  End If.
  If (slot state is Packet Missing) then
    Search DataQ for missing packet
    If (packet is found in DataQ) then
      Remove packet from DataQ
      Place packet in OrderingQ
      Mark slot as Packet Received
      Attempt-Packet-Delivery(slot)
      Update information about packet source
    Else
      Send NACK for packet
      Mark slot as Packet Requested
    End If.
  Else If (slot state is Packet Committed) then
    If (token has been passed enough times to satisfy QoS
        resiliency requirements) then
      Mark slot as Packet Delivered
      Notify application that QoS resiliency has been met
      Update information about packet source
    End If.
  Else If (slot state is Packet Requested) then
    Search DataQ for missing packet
    If (packet is found in DataQ) then
      Remove packet from DataQ
      Place packet in OrderingQ
      Mark slot as Packet Received
      Attempt-Packet-Delivery(slot)
      Update information about packet source
    End If.
  Else If (slot state is Packet Received) then
    Attempt-Packet-Delivery(slot)
    Update information about packet source
  Else If (slot state is Packet Delivered) then
    Update information about packet source
  End If.
End for.
```

```
while (the number of ACK Packets and New Lists Packets in OrderingQ
      is greater than the number of members of the Token Ring) Loop
  DeQueue lowest timestamp and discard packet
End while.
End Update-OrderingQ.
```

As auxillarily algorithm, Attempt-Packet-Delivery, is used by Update-OrderingQ to determine if a packet is deliverable and to deliver the packet to the application. The Attemp-Packet-Delivery algorithm is presented below.

```
Attempt-Packet-Delivery( slot ) :
  If (slot is a Data Packet or Non-Member Data Packet) then
    If (slot packet has QoS equal to Unordered) then
      Commit the packet to the application
      Mark slot as Packet Delivered
    Else If (slot packet has QoS equal to Source Ordered) then
      If (all of the smaller sequence numbers from that
          source have been delivered) then
        Commit the packet to the application
        Mark slot as Packet Delivered
      End If.
    Else If (slot packet has QoS equal to Totally Ordered) then
      If (all of the timestamps smaller than the slots
          timestamps have been delivered) then
        Commit the packet to the application
        Mark slot as Packet Delivered
      End If.
    Else If (slot packet has QoS equal to K Resilient or
              slot packet has QoS equal to Majority
              Resilient or
              slot packet has QoS equal
              to Totally Resilient) then
      If (all of the timestamps smaller than the slots
          timestamps have been delivered) then
        Commit the packet to the application
        Mark slot as Packet Committed
      End If.
    End If.
  Else If (slot is a New List Packet)
    If (all of the timestamps smaller than the slots timestamps have
        been delivered) then
      Commit the New List and notify application
      Mark slot as Packet Delivered
    End If.
  Else If (slot is an ACK Packet) then
    Mark slot as Packet Delivered
  End If.
End Attempt-Packet-Delivery.
```

These two algorithms describe the behavior desired for packets to be ordered and delivered. The last algorithm presented is the Pass-Token algorithm. This algorithm describes how a Token Site is to fill pass the token through either a NL or an ACK. The algorithm is presented below.

```
Pass-Token():
for each (member of the DataQ)
  If (member is a List Change Request Packet and request can
    be granted and packet is eligible) then
    Generate a New List Packet for request
    Send New List Packet
    Exit Loop
  Else If (member is a Data Packet or a Non-Member Data Packet
    and is eligible to be acknowledged) then
    Generate ACK Packet containing as many Data Packets
    and Non-Member Data Packets as are eligible in the DataQ
    Send ACK Packet
    Exit Loop
  End If.
End for.
If (ACK Packet or New List Packet could not be generated) then
  Return to calling routine reporting Token Not Passed
Else
  Return to calling routine reporting Token Passed
End If.
End Pass-Token.
```

For an LCR, Data, or NMD packet to be eligible, the sequence number of the packet must have the expected next value for the source. No strict upper bound is placed on the number of NMD or Data packets an ACK may contain, although differing implementations may impose their own limits.

Delivery of low QoS packets is also checked as the packet arrives and placed into the DataQ. If conditions are such that the packet meets its QoS when it arrives, then the packet may be delivered as well as being placed into the DataQ. Thus when the Update-OrderingQ algorithm pulls the packet out of the DataQ, the slot into which it is placed is to be marked as Packet Delivered instead of Packet Received. This behavior is assumed in the state tables presented below. Another behavior assumed in the preceding algorithms is that the implementation will block packets with resilient QoSs from actual delivery to the application until the conditions meeting their QoS are met. This does mean blocking the delivery of other QoSs (Source and Totally Ordered) that depend on that higher QoS packet.

State Tables and Actions

In the tables presented below each state is shown along with the actions and conditions required to transition into the next state. Assume that events not shown are ignored unless they are specifically discussed below. The different states of RMP operation are:

TS	Token Site State
NTS	Not Token Site State
GP	Getting Packets State
PT	Passing Token State
JR	Joining Ring State
LR	Leaving Ring State
NIR	Not In Ring State
SR	Start Recovery State
CNL	Created New List State
SV	Sent Vote State
ACKNL	ACK New List State
AR	Abort Recovery State

In all states the following behaviors are necessary. All NACK events are handled the same way no matter what the current state is. The default action for NACK responses is to examine the OrderingQ and retransmit the requested timestamped packet (either explicit timestamp or implicit timestamp). NACK policy controls how these are sent, via multicast or unicast, and the timing factors involved. Another issue is generation of NMA packets in response to NMD packets. The default behavior is to have the site which generates the ACK containing the NMD to generate an NMA for that NMD and unicast the NMA to the client. If after this is done, a member of the group sees a duplicate NMD (one that has already been acknowledged and ordered), then the member of the list generates an NMA and unicasts it to the client. The down side of this is that this may result in NMA implosion to the client. This policy may also be changed on an implementation basis to be that the same site always sends the NMA.

It is recommended that implementations differentiate, to the application, NMA packets that hold replies and NMA packets that do not hold replies. The default behavior for a client is to periodically send an NMD to the group (via unicast or multicast) until it receives an NMA (holding no reply) for that NMD, notify the application that the group has received the message, then wait for another NMA that holds a reply. If the reply is in the form of multiple NMA packets, then each should be delivered as they arrive. Missing NMA replies can be requested by sending another NMD (with an increased sequence number), thus causing another message to be delivered to the group. This can be done repeatedly until a client finally gets its reply. Alternatively, a client may mark the Multiple Copies flag of the NMD so that multiple copies of the same NMD will be delivered to the application as they arrive.

Due to the fact that this set of policies does not give any guarantees to the client that the desired QoS was met, the application is fully responsible for the response policy that it will use. For example, if clients built for an application need to know when the QoS is met for the NMD, then the group must send a NMA with a response when that QoS is met and the NMD delivered to the application.

Token Site State Table (current state is TS). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	Token Passed	PT	place packet in DataQ Pass-Token
Data	!Token Passed	TS	place packet in DataQ Pass-Token
NMD	Token Passed	PT	place packet in DataQ Pass-Token
NMD	!Token Passed	TS	place packet in DataQ Pass-Token
LCR	Token Passed	PT	place packet in DataQ Pass-Token
LCR	!Token Passed	TS	place packet in DataQ Pass-Token
ACK	Named Token Site	TS	Unicast Confirm to Source
NL	Named Token Site	TS	Unicast Confirm to Source
Failure	(none)	SR	Multicast RecStart
RecStart	(none)	SV	Unicast RecVote to Reform Site
TPA	(none)	PT	Generate Null ACK Multicast Null ACK
CTPA	(none)	TS	Unicast Confirm to last Token Site

Passing Token State Table (current state is PT). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	(none)	PT	place packet in DataQ Update-OrderingQ
NMD	(none)	PT	place packet in DataQ Update-OrderingQ
LCR	(none)	PT	place packet in DataQ Update-OrderingQ
NL	!named Token Site	NTS	Add NL to OrderingQ Update-OrderingQ
NL	named Token Site	PT	Add NL to OrderingQ Update-OrderingQ
NL	OrderingQ consistent Token passed	TS	Pass-Token
NL	named Token Site	TS	Add NL to OrderingQ Update-OrderingQ
NL	OrderingQ consistent !Token passed	GP	Pass-Token
NL	named Token Site !OrderingQ consistent	GP	Add NL to OrderingQ Update-OrderingQ
ACK	!named Token Site	NTS	Add ACK to OrderingQ Update-OrderingQ
ACK	named Token Site	PT	Add ACK to OrderingQ Update-OrderingQ
ACK	OrderingQ consistent Token passed	TS	Pass-Token
ACK	named Token Site	TS	Add ACK to OrderingQ Update-OrderingQ
ACK	OrderingQ consistent !Token passed	GP	Pass-Token
ACK	named Token Site !OrderingQ consistent	GP	Add ACK to OrderingQ Update-OrderingQ
Conf	Timestamp >= Last token pass Timestamp	NTS	Update-OrderingQ
Failure	(none)	SR	Multicast RecStart
RecStart	(none)	SV	Unicast RecVote to Reform Site

Not Token Site State Table (current state is NTS). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	(none)	NTS	place packet in DataQ Update-OrderingQ
NMD	(none)	NTS	place packet in DataQ Update-OrderingQ
LCR	(none)	NTS	place packet in DataQ Update-OrderingQ
NL	!named Token Site	NTS	Add NL to OrderingQ Update-OrderingQ
NL	named Token Site OrderingQ consistent	PT	Add NL to OrderingQ Update-OrderingQ
NL	Token passed named Token Site OrderingQ consistent	TS	Pass-Token Add NL to OrderingQ Update-OrderingQ
NL	!Token passed named Token Site !OrderingQ consistent	GP	Pass-Token Add NL to OrderingQ Update-OrderingQ
ACK	!named Token Site	NTS	Add ACK to OrderingQ Update-OrderingQ
ACK	named Token Site OrderingQ consistent	PT	Add ACK to OrderingQ Update-OrderingQ
ACK	Token passed named Token Site OrderingQ consistent	TS	Pass-Token Add ACK to OrderingQ Update-OrderingQ
ACK	!Token passed named Token Site !OrderingQ consistent	GP	Pass-Token Add ACK to OrderingQ Update-OrderingQ
Failure	(none)	SR	Multicast RecStart
RecStart	(none)	SV	Unicast RecVote to Reform Site
CommitNL	NL does not contain site	LR	Schedule MandLv

Getting Packets State Table (current state is GP). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	OrderingQ consistent Token passed	PT	place packet in DataQ Update-OrderingQ Pass-Token
Data	OrderingQ consistent !Token passed	TS	place packet in DataQ Update-OrderingQ Pass-Token
Data	!OrderingQ consistent	GP	place packet in DataQ Update-OrderingQ
NMD	OrderingQ consistent Token passed	PT	place packet in DataQ Update-OrderingQ Pass-Token
NMD	OrderingQ consistent !Token passed	TS	place packet in DataQ Update-OrderingQ Pass-Token
NMD	!OrderingQ consistent	GP	place packet in DataQ Update-OrderingQ
LCR	(none)	GP	place packet in DataQ Update-OrderingQ
ACK	OrderingQ consistent Token passed	PT	Add ACK to OrderingQ Update-OrderingQ Pass-Token
ACK	OrderingQ consistent !Token passed	TS	Add ACK to OrderingQ Update-OrderingQ Pass-Token
ACK	!OrderingQ consistent	GP	Add ACK to OrderingQ Update-OrderingQ
NL	OrderingQ consistent Token passed	PT	Add NL to OrderingQ Update-OrderingQ Pass-Token
NL	OrderingQ consistent !Token passed	TS	Add NL to OrderingQ Update-OrderingQ Pass-Token
NL	!OrderingQ consistent	GP	Add NL to OrderingQ Update-OrderingQ
Failure	(none)	SR	Multicast RecStart
RecStart	(none)	SV	Unicast RecVote to Reform Site

Not In Ring State Table (current state is NIR). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
JoinReq	(none)	JR	Multicast LCR to join ring
NMA	NMA holds reply	NIR	Deliver reply to application

Joining Ring State Table (current state is JR). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
NL	named Token Site	TS	Add NL to OrderingQ Update-OrderingQ
Failure	(none)	TS	Generate NL to form own Token Ring

Leaving Ring State Table (current state is LR). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
NL	Exit condition met	NIR	(none)
NL	!Exit condition met	LR	(none)
ACK	Exit condition met	NIR	(none)
ACK	!Exit condition met	LR	(none)
MandLv	(none)	NIR	(none)

NOTE: Exit condition is that a number of Token Passes has occurred in the group equal to the number of people in the group when the site transitioned into LR.

Start Recovery State Table (current state is SR). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	(none)	SR	place packet in DataQ Update-OrderingQ Update SynchTSP
NMD	(none)	SR	place packet in DataQ Update-OrderingQ Update SynchTSP
LCR	(none)	SR	place packet in DataQ Update-OrderingQ
ACK	(none)	SR	Add ACK to OrderingQ Update-OrderingQ
NL	(none)	SR	Update SynchTSP and MaxTSP Add NL to OrderingQ Update-OrderingQ Update SynchTSP and MaxTSP
Failure	packet is RecStart	CNL	Create New List Multicast New List
Failure	!only site packet is RecStart	NIR	Create New List Commit New List
Failure	only site List is invalid packet is RecStart	TS	Create New List Commit New List Multicast New List
Failure	only site List is valid		
RecVote	incorrect Info	AR	Multicast RecAbort
RecVote	Vote MaxTSP > MaxTSP	SR	Update MaxTSP
RecVote	Vote MaxTSP <= MaxTSP	SR	Update source vote
RecVote	OrderingQ consistent Vote from each site Each Voter synched	CNL	Create New List Multicast New List
RecAbort	correct Info	AR	(none)
RecStart	incorrect Info	AR	Multicast RecAbort

NOTE: incorrect Info means that the Token Ring Version, New Token Ring ID, or Reform Site information in the packet is incorrect. Any update to the value of MaxTSP or SynchTSP for the Reform Site prompts a restart of the RecStart packet. Each Voter is synched if the vote from that voter has its SynchTSP set to the MaxTSP of the current reformation.

Created New List State Table (current state is CNL). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	(none)	CNL	place packet in DataQ Update-OrderingQ
NMD	(none)	CNL	place packet in DataQ Update-OrderingQ
LCR	(none)	CNL	place packet in DataQ Update-OrderingQ
RecACKNL	incorrect Info	AR	Multicast RecAbort
RecACKNL	!All sites have ACKed	CNL	Mark source as ACKed
RecACKNL	All sites have ACKed List is valid	PT	Add NL to OrderingQ Commit NL Multicast Null ACK
RecACKNL	All sites have ACKed List is invalid	NIR	Add NL to OrderingQ Commit NL
Failure	packet is NL List is valid	AR	Multicast RecAbort
Failure	packet is NL List is invalid	NIR	Add NL to OrderingQ Commit NL
RecAbort	correct Info	AR	(none)
RecStart	incorrect Info	AR	Multicast RecAbort
RecVote	correct Info	CNL	Unicast Reformation NL to voting site

NOTE: Reformation NL is generated in the SR state. This NL is not added to the OrderingQ or committed by the Reform Site until it is positive that all the Slave Sites have it. This is shown as having ACKs from all those involved in Reformation. Valid lists are lists that pass the Minimum Size Requirements as set forth by the members. Invalid lists fail to meet these requirements.

ACK and NL packets that are not duplicates arriving in this state are to cause warnings. The ring should not be sending them because the synchronization point for the group has already been chosen and should not be changed.

Sent Vote State Table (current state is SV). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	(none)	SV	place packet in DataQ Update-OrderingQ Update RecVote
NMD	(none)	SV	place packet in DataQ Update-OrderingQ Update RecVote
LCR	(none)	SV	place packet in DataQ Update-OrderingQ
ACK	(none)	SV	Add ACK to OrderingQ Update-OrderingQ Update RecVote
NL	!Reformation NL	SV	Add ACK to OrderingQ Update-OrderingQ Update RecVote
NL	Reformation NL !site in list	NIR	(none)
NL	Reformation NL List is invalid	NIR	Unicast RecACKNL to Reform Site Commit NL
NL	Reformation NL List is valid	ACKNL	Unicast RecACKNL to Reform Site
RecStart	correct Info	SV	Unicast RecVote to Reform Site
RecStart	incorrect Info	AR	Multicast RecAbort
Failure	packet is RecVote	AR	Multicast RecAbort
RecAbort	correct Info	AR	(none)

NOTE: Each time a Slave Site receives a RecStart from the Reform Site, the Slave Site is to restart its retransmission schedule for its RecVote. This is needed so that updates to the MaxTSP and SynchTSP of the reformation do not cause a Slave Site to prematurely initiate abortion of a reformation. Every time a Slave Site updates its RecVote, the site must also restart its retransmit schedule on the RecVote packet. A Reformation NL is a New List that meets the following criteria: (1) Version number is correct with current Reformation, (2) The current Token Site and the next Token Site are the current Reform Site, (3) The Timestamp of the NL must be in the range SynchTSP+1 to MaxTSP+1 of the current Reformation, (4) The operation type of the NL must be a reformation operation, and (5) The new TRID of the NL must be equal to the TRID of the reformation.

It is practically impossible at this step of reformation to tell the

difference between a viable reformation NL and an incorrect one.

ACK New List State Table (current state is ACKNL). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	(none)	ACKNL	place packet in DataQ Update-OrderingQ
NMD	(none)	ACKNL	place packet in DataQ Update-OrderingQ
LCR	(none)	ACKNL	place packet in DataQ Update-OrderingQ
NL	Timestamp < Reformation NL	ACKNL	Add NL to OrderingQ Update-OrderingQ
NL	Reformation NL	ACKNL	Unicast RecACKNL to Reform Site
NL	!Reformation NL incorrect Info	AR	Multicast RecAbort
NL	!Reformation NL Timestamp > Reformation NL !named Token Site	NTS	Add Reformation NL to OrderingQ Commit Reformation NL Add NL to OrderingQ Update-OrderingQ
NL	!Reformation NL Timestamp > Reformation NL named Token Site OrderingQ consistent Token passed	PT	Add Reformation NL to OrderingQ Commit Reformation NL Add NL to OrderingQ Update-OrderingQ Pass-Token
NL	!Reformation NL Timestamp > Reformation NL named Token Site OrderingQ consistent !Token passed	TS	Add Reformation NL to OrderingQ Commit Reformation NL Add NL to OrderingQ Update-OrderingQ Pass-Token
NL	!Reformation NL Timestamp > Reformation NL named Token Site !OrderingQ consistent	GP	Add Reformation NL to OrderingQ Commit Reformation NL Add NL to OrderingQ Update-OrderingQ
ACK	Timestamp < Reformation NL	ACKNL	Add ACK to OrderingQ Update-OrderingQ
ACK	Timestamp > Reformation NL !named Token Site	NTS	Add Reformation NL to OrderingQ Commit Reformation NL Add ACK to OrderingQ Update-OrderingQ
ACK	Timestamp >	PT	Add Reformation NL

	Reformation NL		to OrderingQ
	named Token Site		Commit Reformation NL
	OrderingQ consistent		Add ACK to OrderingQ
	Token passed		Update-OrderingQ
			Pass-Token
ACK	Timestamp >	TS	Add Reformation NL
	Reformation NL		to OrderingQ
	named Token Site		Commit Reformation NL
	OrderingQ consistent		Add ACK to OrderingQ
	!Token passed		Update-OrderingQ
			Pass-Token
ACK	Timestamp >	GP	Add Reformation NL
	Reformation NL		to OrderingQ
	named Token Site		Commit Reformation NL
	!OrderingQ consistent		Add ACK to OrderingQ
			Update-OrderingQ
Failure	packet is RecACKNL	AR	Multicast RecAbort
RecAbort	correct Info	AR	(none)
RecStart	incorrect Info	AR	Multicast RecAbort

NOTE:

Abort Recovery State Table (current state is AR). Events not mentioned have no action and cause no transition. The sequence of actions are executed before the conditions are checked.

Event	Condition(s)	Next State	Action(s)
Data	(none)	AR	place packet in DataQ Update-OrderingQ
NMD	(none)	AR	place packet in DataQ Update-OrderingQ
LCR	(none)	AR	place packet in DataQ Update-OrderingQ
NL	Old Version	AR	Add NL to OrderingQ Update-OrderingQ
NL	Reformation NL Version is of last attempt	AR	Multicast RecAbort
ACK	(none)	AR	Add ACK to OrderingQ Update-OrderingQ
RTA	(none)	SR	Multicast RecStart
RecStart	correct Version	SV	Unicast RecVote to Reform Site
RecStart	incorrect Version	AR	Multicast RecAbort
RecVote	incorrect Version	AR	Multicast RecAbort
RecACKNL	incorrect Version	AR	Multicast RecAbort

NOTE: A correct version at this stage is a version greater than the last known version. Fill the RecAbort packet in with data in the incoming packet. A NL that qualifies as a Reformation NL is any NL that has an operation type dealing with Reformation and has a version higher than the last known version.

Authors' Addresses:

Brian Whetten
University of California Berkeley
Berkeley, CA
Email: whetten@tenet.cs.berkeley.edu

Todd Montgomery
West Virginia University
Morgantown, WV
Email: tmont@cerc.wvu.edu

John R. Callahan
West Virginia University
Morgantown, WV

Email: callahan@cerc.wvu.edu

